MCGINN & GIBB, PLLC
A PROFESSIONAL LIMITED LIABILITY COMPANY
PATENTS, TRADEMARKS, COPYRIGHTS, AND INTELLECTUAL PROPERTY LAW
8321 OLD COURTHOUSE ROAD, SUITE 200
VIENNA, VIRGINIA 22182-3817
TELEPHONE (703) 761-4100
FACSIMILE (703) 761-2375; (703) 761-2376

APPLICATION FOR UNITED STATES LETTERS PATENT

APPLICANT: CHEN, ET AL.

FOR: METHOD AND STRUCTURE FOR MONITORING MOVING OBJECTS

DOCKET NO.: YOR920030164US1

METHOD AND STRUCTURE FOR MONITORING MOVING OBJECTS

Cross-Reference to Related Applications

The present Application is related to the following co-pending application:

U.S. Patent Application No. 10/ ___,___, filed on _____, to Chen et al., entitled "System and Method for Monitoring Events Against Continual Range Queries", having IBM Docket YOR920030165US1, assigned to the present assignee, and incorporated herein by reference; and

U.S. Patent Application No. 10/ ___,___, filed on _____, to Chen et al, entitled "System and Method for Indexing Queries, Rules and Subscriptions", having IBM Docket YOR920030265US1,

both assigned to the present assignee, and both incorporated herein by reference.

DESCRIPTION

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention generally relates to providing location-aware (or location-dependent) services in a mobile environment. More specifically, a query indexing system and method that allow the user of this invention to periodically and incrementally locate all the moving objects inside the boundaries of continual range queries. "Incrementally" may be

defined as skipping certain query evaluations by filtering out a subset of moving objects based on the relative movements from the last positions with respect to query boundaries.

Description of the Related Art

Location-aware (or location-dependent) services have become possible and emerging recently, thanks to the advances in mobile computing and location-sensing technologies, such as the global positioning system (GPS). These services can improve the quality of life by adding location-awareness to many objects of interest, such as humans, taxi cabs, ambulances, laptops, airplanes, trains and cargo ships. Basically, any object that moves can be equipped with location-awareness and monitored.

One of the most fundamental technical problems for providing location-aware services is to monitor and provide fast answers to continual range queries that locate the moving objects inside the query boundaries. With the results of continual range queries readily available, various kinds of location-aware services can be provided. For example, a taxi cab service company can dispatch a taxi cab to a customer at a specific location using the result from a continual range query such as: "Finding all the taxi cabs currently located within five blocks away from that specific location." The range queries are termed "continual" because they are repeatedly evaluated in order to provide the most up-to-date answers as objects continuously to move around. Each range query specifies the boundaries of a region.

An efficient index is usually needed for monitoring numerous continual range queries over a large number of moving objects. There are generally two different indexing approaches. One is to index the moving objects, and the other is to index the range queries.

Various attempts have been proposed to build an index on the moving objects, such as the following:

- "Indexing moving objects," by P.K. Agarwal et al., in *Proceedings of ACM Symposium on Principles of Database Systems*, 2000;

- "On indexing mobile objects," by G. Kollios et al., in *Proceedings of ACM Symposium on Principles of Database Systems*, 1999;

- "Novel approaches to the indexing of moving object trajectories," by D. Pfoser et al., in *Proceedings of Very Large Data Bases*, 2000; and

- "Indexing the positions of continuously moving objects," by S. Saltenis et al., in *Proceedings of ACM SIGMOD,* 2000.

However, because objects may continuously move in unpredictable speed and direction, it is very difficult to maintain an effective index on the moving objects. Changes in object locations demand updates to the object index, greatly degrading its performance. As a result, certain constraints are usually placed on the moving speed or direction of an object, significantly limiting the applicability of the object index.

In contrast, it is more effective to build an index on the range queries because continual range queries change less frequently. With query indexing, the problem of monitoring continual range queries becomes the following: Given a set of range queries and a set of moving objects, continually determine the set of objects that are located inside the boundaries of each range query.

The query index is used to quickly retrieve all the range queries that cover a given object. Periodically, all the range queries are reevaluated by identifying all the objects

covered by each range query taking into account their latest locations. In order for the results to be useful, the time period between two consecutive reevaluations must be short. As a result, the time for a query reevaluation must be as brief as possible. Moreover, it is important that a query indexing method can take advantage of incremental changes in object locations because some of the objects may not have moved outside a query boundaries since the last reevaluation. This invention discloses an effective query indexing system and method for monitoring continual range queries over moving objects.

Query indexing was not used in the moving object environment until recently. In "Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects," *IEEE Transactions on Computers*, 51:1124-1140, Oct. 2002, S. Prabhakar et al. proposed a query indexing method using an R-tree. However, in order to avoid excessive location updates, a safe region for each mobile object was defined. Unfortunately, determining a safe region requires intensive computation. Moreover, a constraint is imposed on the maximum velocity of a moving object to compute the safe regions.

In "Efficient evaluation of continuous range queries on moving objects," *Proceedings of 13th International Conference on Database and Expert Systems Applications*, 2002, D. V. Kalashnikov et al. proposed a grid cell-based query indexing method, which was shown to outperform an R-tree-based query index. The monitoring area is partitioned into non-overlapping grid cells. Each cell maintains two lists: full and partial. The full list stores the IDs of queries that completely cover the cell, while the partial list maintains the IDs of queries that partially intersect with the cell.

However, the need to use the partial lists has a significant drawback. The fact that an object is inside a cell does not imply that it is inside the boundaries of a query stored in the partial list of the cell. This forces continual comparisons of object locations against the boundaries of queries. As a result, it cannot take advantage of the incremental changes in object locations. Even if an object has not moved outside a cell, boundary comparisons against all the queries stored in a partial list are still needed.

As a result, a need has been recognized to have a better and more effective query indexing method that:

(1) does not impose any limit on the moving speed or direction of an object, and

(2) can take advantage of incremental changes in object locations.

Those skilled in the art will appreciate that various query indexing methods have been proposed to efficiently matching events in the context of predicate matching (e.g., E. Hanson et al., "Selection predicate indexing for active databases using interval skip lists," *Information Systems*, 21(3):269-298, 1996) and pub/sub (e.g., M. K. Aguilera et al., "Matching events in a content-based subscription system," *Proceedings of Symposium on Principles of Distributed Computing*, 1999; F. Fabret et al, "Filtering algorithms and implementation for very fast publish/subscribe systems," *Proceedings of the ACM SIGMOD*, 2001).

However, these query indexing methods are mostly based on equality predicates, not range predicates. Thus, they are not generally applicable for the evaluation of continual range queries over moving objects.

Those skilled in the art will also appreciate that, although range queries can be treated as spatial objects such as rectangles, traditional spatial indexing methods, such as R-trees (e.g., A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Proceedings of ACM SIGMOD*, 1984; V. Gaede et al., "Multidimensional access methods," *ACM Computing Surveys*, 30(2):170-231, 1998), are not effective for monitoring moving objects because they are mostly disk-based indexing methods.

They are generally too slow to be effective for monitoring continual range queries over a large number of moving objects. Moreover, the performance of an R-tree quickly degenerates when the ranges of continual queries start to overlap one another (V. Gaede et al., "Multidimensional access methods," *ACM Computing Surveys*, 30(2):170-231, 1998; E. Hanson et al., "Selection predicate indexing for active databases using interval skip lists," Information Systems, 21(3):269-298, 1996).

SUMMARY OF THE INVENTION

In view of the foregoing problems, drawbacks, and disadvantages of the conventional systems, it is an exemplary feature of the present invention to provide a query indexing structure (and method) which allows the user of this invention to periodically and incrementally locate all the moving objects inside the boundaries of continual range queries. In this context, "incrementally" means skipping certain query evaluations by filtering out a subset of moving objects based on the relative movements since the last positions with respect to query boundaries.

It is, therefore, an exemplary purpose of the present invention to provide a structure and method for a query indexing structure (and method) which allows the users of this invention to periodically and incrementally locate all the moving objects inside the boundaries of continual range queries.

Hence, in a first aspect of the present invention, described herein is a method and structure that monitors continual queries over moving objects, identifying a query region in a digital format. Each query region is strictly covered by at least one shingle, such that each query region is completely covered by the at least one shingle and no section of any of the at least one shingle falls outside the query region.

In a second aspect of the present invention, described herein is a service based on monitoring continual queries over moving objects, wherein the service includes at least one of: providing a monitoring of moving objects against continual queries, using the just-described method; providing a result of the monitoring using the method; and using a result of the monitoring using the method.

In a third aspect of the present invention, described herein is a signal-bearing medium tangibly embodying a program of machine-readable instructions executable by a digital processing apparatus to perform the above-described method of monitoring continual queries over moving objects.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other exemplary features, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

Figure 1 shows the system block diagram of a mobile environment 100 where a plurality of continual range queries can be monitored against a large number of moving objects;

Figure 2 shows a summary flow chart 200 of an exemplary preferred embodiment of the present invention;

Figure 3 shows the concept 300 of virtual shingles used for query indexing according to the present invention;

Figure 4 shows the concept 400 of strict covering 401 versus loose covering 402 of the area shown covered by diagonal lines;

Figures 5 and 6 show an example 500 of covering a range query 501, using a concept of strip rectangles and one or more virtual shingles 503-504, 602-604, some of which may overlap each other;

Figure 7 shows an example 700 of a shingle-based query indexing;

Figure 8 shows an example 800 of finding the covering shingles for any object location;

Figure 9 shows the flow chart 900 of an exemplary algorithm for enumerating all the covering shingles for an object location;

Figure 10 shows an example 1000 of incremental query evaluation once an object moves to a new location;

Figure 11 shows the flow chart 1100 of the algorithm for incrementally evaluating queries by skipping certain computations;

Figure 12 illustrates an exemplary hardware/information handling system 1200 for incorporating the present invention therein; and

Figure 13 illustrates a signal bearing medium 1300 (e.g., storage medium) for storing steps of a program of a method according to the present invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT OF THE INVENTION

Referring now to the drawings, and more particularly to Figure 1, a preferred embodiment of the present invention will now be described. Figure 1 shows an exemplary system block diagram 100 of a mobile environment where a plurality of continual range queries 110-114 can be monitored against a large number of moving objects, exemplarily shown as objects 120-128. A satellite 101 can be used to relay and broadcast data to the receivers 130, 131 on the ground. On the ground, are many mobile objects 120-128, along with regional computer servers 150,151, which servers are equipped with wireless receivers 130, 131.

The range queries 110-114 are shown as rectangles in a two dimensional space in this exemplary preferred embodiment. However, those skilled in the art will appreciate that they can be of other shapes, such as circles in a 2D space, cubes or spheres in a 3D space.

These continual range queries 110-114 specify the regions that are to be monitored. The query results are all the objects that are located inside individual query regions. The boundaries of these range queries are maintained by one or more regional computer servers 150, 151. These regional servers are connected via a communication network 140.

The moving objects 120-128 are equipped with location-awareness devices, such as Global Positioning System (GPS) devices. The locations of moving objects 120-128 are also maintained by one or more regional computer servers 150, 151. Changes in object locations are reported to the computer servers 150, 151 by the moving objects 120-128, typically via wireless links 130, 131.

Figure 2 shows in summary format 200 the exemplary shingle-based query indexing (SQI) method disclosed herein for effectively monitoring numerous continual range queries over a large number of moving objects. A shingle is a tile-like object, not necessarily of rectangular shape, that is conventionally laid in overlapping rows to cover an area. In the context of the present invention, a shingle may be defined as a digital representation of a tile-like object laid to cover a digital representation of an area (e.g., a geographical area), without necessarily being laid in overlapping rows.

A core idea of the present invention is to use one or more shingles to cover each range query. In contrast to cells, the shingles of the present invention may overlap with one another, similar to shingles used to cover the roof of a house. With SQI, there is no

ambiguity as to whether or not a point is covered by a query if it falls inside one of the shingles that cover the query. Hence, incremental reevaluation approach can be easily pursued.

Computation can be saved for those objects that have not moved outside a shingle since the last reevaluation. Since an object can be covered by more than one shingles, an exemplary efficient algorithm to determine the set of shingles that cover an object is first provided. Incremental reevaluation is accomplished by using the difference between the sets of covering shingles for the new location and old location.

To implement an SQI, a set of virtual shingles are predefined. Each virtual shingle has a unique bottom-left corner (*a, b*), width and height. A shingle stays virtual until it is used to cover a range query. At that time, it is considered as having been activated.

In steps 201 and 202, when a range query is to be inserted, a set of virtual shingles to cover the range query is first found. Then, in step 203, the query ID is inserted into each of the ID lists associated with the covering shingles. There are many ways that could be used to cover a query with shingles.

In a preferred exemplary embodiment, a simple yet systematic approach is presented. First, a strip rectangle of height k from the bottom of the query rectangle is drawn and moved upwards. For the last strip, the bottom of the strip is allowed to overlap with a previous one. For each strip rectangle, virtual shingles are used to cover it from the left side, moving towards the right side. For the last shingle, the right side is lined up and the left side of the shingle is allowed to overlap with a previous covering shingle.

To search all the queries that cover a given object, the covering shingles of the object are first computed and then, from these covering shingles, are all the queries that cover the object. An exemplary efficient algorithm to enumerate all the covering shingles for any given object location is presented below. This enumeration algorithm is made possible by two important properties: constant size, and identical gap pattern. The number of covering shingles is the same for all object locations. If the IDs of the covering shingles are sorted in an increasing order, the gap between any two shingles of matching positions is identical for any two locations.

A complete query reevaluation is done as follows. For each object, the covering shingles for it are found, and then all the queries that cover these shingles are found. The object ID is then stored in the object lists associated with these queries. At the end, as shown in step 204, the object lists are the query results.

However, incremental reevaluation is presented to save computation, as follows. For incremental reevaluation, the covering shingles for the new location and the old location are first computed. For all the covering shingles of the new location, but not of the old, an instance of the object is inserted into the object lists pointed to by the queries that cover these shingles. This accounts for the case that the object has moved into these shingles.

For all the covering shingles of the old location, but not of the new location, an instance of the object is deleted from the object lists pointed to by the queries that cover these shingles. This accounts for the case that the object has moved out of these shingles. For any covering shingle that is both of the new location and the old location, nothing needs to be

done. It accounts for the case that the object has remained inside the boundaries of these shingles.

The system and method for monitoring continual range queries against moving objects consist mainly of an effective query indexing method. With this query index, periodically each new object location is used to find all the queries that cover it. The object ID is then stored into an object list (*OL*) associated each of the covering queries.

After all object locations are processed, a complete new set of query results is available. In other words, object list *OL*(*q*) contains all the objects currently located inside the boundaries of query *q*. The data structure for the query index and its associated operations are stored as computer executable programs in one or more of the regional servers 150,151 in Figure 1. The results of these continual range queries can be used to provide various kinds of location-aware services.

The query index of the present invention is based on virtual shingles. Figure 3 exemplarily shows the concept 300 of virtual shingles 301-305. Virtual shingles are covering rectangles. Virtual shingles can overlap with each other. There are tradeoff in terms of storage and performance using overlapping shingles. These shingles remain virtual until they are used to cover a range query, at which point, they become activated. It is noted that each shingle 301-305 shown in Figure 3 is 4 x 4 and that the shingle size remains constant in the exemplary preferred embodiment.

Additionally, each virtual shingle 301-305 has a unique ID which can be computed systematically, as follows. Assuming that the monitoring region is a rectangle defined by *Rx*

and *Ry*, then the identification (ID) of a virtual shingle, whose bottom-left corner is located at coordinate (a,b), can exemplarily be computed as: $s(a,b) = b\ R_X + a$.

Thus, in Figure 3, wherein monitoring region 306 has size Rx =16 and Ry = 12, for virtual shingle 304, having bottom-left corner at coordinate (8,8), the ID for shingle 304 is $s(8,8) = 8 \times 16 + 8 = 136$. Similarly, the ID for shingle 305 is $s(12,8) = 8 \times 16 + 12 = 140$.

Before a range query can be inserted into the query index, one or more virtual shingles are first found such that the range query is strictly covered. Here, "strict covering" means that the entire query region is completely covered by one or more virtual shingles and that none of these covering shingles is outside the boundaries of the query.

Figure 4 illustrates the concept 400 of "strict covering" 401 versus "loose covering" 402. The difference is whether any of the covering shingles exceed the query boundaries. That is, in the case of strict covering shown at the top 401 of the figure, covering shingles 404, 405 completely covers the query boundaries and they remain inside the query region, shown in the figure as being the area 403 covered by diagonal lines. In contrast, as shown in the lower portion 402 of the figure, for the shingles 406, 407 covering region 403, at least one shingle (e.g., 407) exceeds the query boundaries in the loosely covering case (e.g., region 408).

Strict covering has an important property that any point covered by a shingle is guaranteed to be also covered by the query. Without strict covering, this property does not exist. For example, a point in region 408 covered by shingle 407 in Figure 4 is not necessarily covered by the query. However, any point covered by shingle 404 or 405 is covered by the query.

There are many possible ways that a query could be strictly covered with virtual shingles. In this exemplary embodiment, one possible simple and systematic approach is described in reference to Figures 5 and 6.

These two figures 500, 600 show an example of covering a range query 501 with one or more virtual shingles, wherein some of them may overlap with one another. For the description of this preferred embodiment, virtual shingles are assumed to be squares with the same size of k x k (e.g., 4 x 4). It should be apparent that the virtual shingles could be of different sizes and shapes, such as rectangles, from that shown.

Moreover, it is assumed that k is chosen such that it is smaller than or equal to the dimension of the smallest range queries. Namely, all the range queries can be fully covered by one or more virtual shingles. Thus, in Figures 5 and 6, k is exemplarily chosen to be 4.

To cover a given range query 501 (e.g., the area covered by diagonal lines), first, as shown in Figure 5, a strip rectangle 502, having height equal to the height of a virtual shingle (e.g., shingle 503), is generated from the bottom left corner of the query rectangle 501. The strip rectangle 502 can be identified as (3,3,11,4), wherein the symbology (a,b,c,d) of the strip rectangle has the following meaning: (a,b) is the coordinates of the lower left corner of the strip rectangle; c is the length of the strip rectangle; and d is the height of the strip rectangle.

As shown in Figure 6, the strip rectangle generation then moves upwards to become strip rectangle 601, which would have identification (3,5,11,4). For the final strip rectangle (e.g., strip rectangle 601 in this example), the bottom of the strip rectangle 601 is allowed to

overlap with a previous strip rectangle 502. It should be apparent that this overlap of strip rectangles provides strict covering of the range query 501.

Next, for each strip rectangle 502, 601, virtual shingles (e.g., 503-505 and 602-604) are used to cover the strip rectangle, exemplarily starting from the left side and moving towards the right side. For the last shingle (e.g., shingles 505, 604) in each strip rectangle, the right edges of the strip rectangle and the shingle are aligned, thereby allowing the left side of the last shingle to overlap with a previous covering shingle and achieve strict covering of the strip rectangle.

Thus, in Figure 5, three virtual shingles 503-505 are used to cover the first strip rectangle 502 drawn on the lower part of range query 501. Similarly, in Figure 6, three virtual shingles 602-604 are used to cover the second strip rectangle 601 drawn on the upper part of the query 501. Those skilled in the art will appreciate that there are many other ways to completely cover a range query with one or more shingles. It should also be apparent that the overlap of the strip rectangles 502, 601 and the overlap of final shingles 505,604 provide strict covering of the range query 501.

After finding all the covering shingles, a query ID is inserted into ID lists associated with all the covering shingles. Figure 7 shows one possible shingle-based query indexing scheme 700. Essentially, the query index maintains a mapping 701 from each virtual shingle s_k to a list 702 of IDs of queries q_i. That is, list 702 is an array of pointers to the queries 703, 704. Let $QL(s)$ denote the set of query IDs associated with a virtual shingle s. It contains all the queries that s covers. Hence, if a virtual shingle is not used to cover any range query, then the corresponding ID list is empty. As an example, in Figure 7, queries q_1-q_4 are

covered completely by various virtual shingles. Some of them overlap with each other. The query IDs are stored in the corresponding ID lists so that, for example, pointers 703, 704 point respectively to query q_2 for shingle S_i and query q_3 for S_j.

To search all the queries that cover an object location, all the virtual covering shingles must first be found. Let $CS^V(o)$ denote the set of virtual covering shingles for object o (e.g., object O_1, O_2 in Figure 7). From these virtual covering shingles, all the queries stored in the corresponding $QL(s)$ (e.g., 703, 704) must be found for every $s \in CS^V(o)$.

Figure 8 shows an exemplary method 800 of finding all the covering shingles for an object o whose location is at (x, y), where $a < x < a + 1$, and $b < y < b + 1$. All the covering shingles can be systematically enumerated from Figure 8, as follows: their bottom-left corners are located in the shaded area 801. This shaded area 801 is defined by the intersection of two shingles 802, 803 whose upper-right corners are at (a, b) and $(a+1, b+1)$, respectively. In total, there are 16 such possible covering shingles as shown in Figure 8 (e.g., if each point in the shaded area 801 had an associated shingle). The smallest ID of these covering shingle is $s(a+1-k, b+1-k)$, labeled in Figure 8 as shingle 803. The largest ID is $s(a, b)$, labeled in Figure 8 as shingle 804.

Figure 9 shows an exemplary flow chart 900 for systematically enumerating the set of covering shingles, $CS^V(o)$, for an object o whose location is at (x, y), where $a < x < a + 1$, and $b < y < b + 1$. First, in step 901, the covering shingle set is initialized as an empty set and J is initialized to be $b+1-k$. Then, in step 902, it is checked if $(J > b)$ is true. If yes, in step 909, the routine ends and $CS^V(o)$ is returned.

If not, then in step 903, it is further checked if ($J >= 0$) is true. If not, in step 910, J is incremented by 1. Otherwise, in step 904, I is initialized to be $a+1-k$. In step 905, it is checked if ($I > a$) is true. If yes, in step 910, 1 is added to J. If not, in step 906, it is further checked if ($I >= 0$) is true. If yes, in step 907, the shingle s(I, J) is added to $CS^{V}(o)$. In step 908, 1 is added to I. In step 906, if ($I < 0$), the routine skips to step 908. After I is incremented in step 908, the routine returns to step 905.

Those skilled in the art will appreciate that the size of the covering shingle set of any object location remains constant so long as that location is not in the boundary regions. The boundary regions are defined by $0 \leq x < k$, $Rx - k \leq x < Rx$, $0 \leq y < k$, or $Ry - k \leq x < Ry$. This can easily be verified by moving around the object in Figure 8. The relative positions of the covering shingles may change, but the size of the shaded area 801 remains the same.

The concept illustrated by shaded area 801 in Figure 8 can be utilized in determining which shingles are involved when an object moves. That is, given the exemplary procedure for enumerating all the covering shingles for an object location, an exemplary incremental query reevaluation is illustrated by Figure 10. This figure shows an example of saving certain query evaluation when an object moves from one location to another, exemplarily shown as location L1 to location L2. In general, when an object moves, the set of covering shingles for the new location $CS^{V}_{new}(o)$ and that for the old location $CS^{V}_{old}(o)$ may have overlapped. Namely, there are certain shingles that belong to both sets.

For those covering shingles belonging to both sets, no computation is needed. This is because the moving object has not moved outside the boundaries of these covering shingles. However, for those shingles that belong to $CS^{V}_{new}(o)$, but not to $CS^{V}_{old}(o)$, the object ID needs

to be inserted into the corresponding queries that those shingles cover. On the other hand, for those shingles that belong to $CS^{V}_{old}(o)$, but not to $CS^{V}_{new}(o)$, the object ID needs to be deleted from the corresponding queries covered by those shingles.

Figure 10 shows the regions of interest as an object from location L1 to L2, given the environment shown in Figure 8. That is, in Figure 10, the location L1 corresponds to location (x,y) in Figure 8. New location L2 in Figure 10 is approximately one unit away. Therefore, it should be intuitive that the 4 x 4 shaded area 801 in Figure 8 will now shift according to the new location L2. This shift in location of the 4 x 4 shaded area is shown in Figure 10, except that the 4 x 4 area can now be broken down into three areas.

In the first area 1001, this is the 3 x 3 square that is shared by the 4 x 4 area 801 shown in Figure 8 and the new 4 x 4 area corresponding to the new location L2. In this 3 x 3 square 1001 are represented all covering shingles where no action is needed for the object identification list.

Area 1002 is the section of the 4 x 4 shaded area 801 that has been vacated by location to L2. This area 1002 represents those covering shingles for which a deletion in the object identification list needs to be executed.

Finally, area 1003 in Figure 10 represents the newly-added section of the 4 x 4 square. That is, in area 1003 is represented those covering shingle for which an insertion in the object identification list needs to be executed.

Therefore, based on the concepts shown in Figure 10, a flow chart 1100 of an exemplary routine for an incremental reevaluation of queries by filtering out certain computations is shown in Figure 11. The query reevaluation is done for every object in O,

which is the set of all moving objects. When every object has been examined (e.g., steps 1101, 1102, and 1107), the routine stops in step 1103.

First, for each object, in step 1104, the set of covering shingles for the new and old locations are first computed: $CS^V_{new}(o)$ and $CS^V_{old}(o)$. Then, in step 1105, $\forall s_k \in CS^V_{new}(o) - CS^V_{old}(o)$, an instance of o is inserted into $OL(q)$, $\forall q \in QL(s_k)$. After that, in step 1106, $\forall s_j \in CS^V_{new}(o) - CS^V_{old}(o)$, an instance of o is deleted from $OL(q)$, $\forall q \in QL(s_j)$. It is noted that, $\forall s \in CS^V_{new}(o) \cap CS^V_{old}(o)$, no action is needed.

Those skilled in the art will appreciate that range queries may move around, similar to the objects. In such cases, the old range queries are removed from the query index first and the new range queries are inserted. Then, the objects are similarly used to find all the covering range queries in order to periodically reevaluate the query results.

It should also be apparent that various kinds of services can be provided based on the system and method disclosed in this current invention. As a first example, a service can be offered to a taxi cab company to dispatch a taxi to a customer asking for such a service. The service provider would monitor the taxi cabs as moving objects. Their locations are reported to the service providers periodically.

One or many taxi stops, such as hotels or restaurants, and the ranges surrounding these stops could represent range queries of the present invention. The service provider would repeatedly maintain the taxi cabs currently within the range queries using the system and method disclosed in the present invention. Such query results can be provided to the taxi cab company to appropriately dispatch taxi cabs to customers who are calling from one of the stops.

In another example, a service based on the system and method of the current invention could be provided to one or more stores for the purpose of allowing these stores to dynamically send electronic coupons to customers with cell phones or PDAs who are currently approaching the stores. In this example, the stores and their desired ranges are the queries. Customers with cell phones or PDAs are moving objects. A service provider can monitor these moving objects against the stores and their range queries. A store owner who wants to send e-coupons to the customers currently around its store can take advantage of such a service.

However, it should be apparent that these examples above are only two possible applications of the present invention and are not intended as limiting the present invention in any way. The present invention provides a computerized technique of monitoring moving objects and, returning back to the block diagram of Figure 1, will be of interest to any of the levels of users or implementation shown in Figure 1. Thus, a consumer of the present invention could be considered as the end user represented as the moving objects 120-128 being tracked by the technique described by the present invention, as an operator of a service (e.g., taxi company) or store (e.g., for electronic coupon distribution) that utilizes the result of the query, or as the service provider 150, 151 that directly executes the present invention either for its own use or for forwarding to clients such as a taxi company. Under certain conditions, it is possible that the owner/operator of the object tracking system (e.g., shown as satellite 101), the wireless reception facility 130, 131, or even the computer network 140 might be considered as the consumer of the present invention.

Exemplary Hardware Implementation

Figure 12 illustrates a typical hardware configuration of an information handling/computer system 1200 in accordance with the invention and which preferably has at least one processor or central processing unit (CPU) 1211.

The CPUs 1211 are interconnected via a system bus 1212 to a random access memory (RAM) 1214, read-only memory (ROM) 1216, input/output (I/O) adapter 1218 (for connecting peripheral devices such as disk units 1221 and tape drives 1240 to the bus 1212), user interface adapter 1222 (for connecting a keyboard 1224, mouse 1226, speaker 1228, microphone 1232, and/or other user interface device to the bus 1212), a communication adapter 1234 for connecting an information handling system to a data processing network, the Internet, an Intranet, a personal area network (PAN), etc., and a display adapter 1236 for connecting the bus 1212 to a display device 1238 and/or printer 1239 (e.g., a digital printer or the like).

In addition to the hardware/software environment described above, a different aspect of the invention includes a computer-implemented method for performing the above method. As an example, this method may be implemented in the particular environment discussed above.

Such a method may be implemented, for example, by operating a computer, as embodied by a digital data processing apparatus, to execute a sequence of machine-readable instructions. These instructions may reside in various types of signal-bearing media.

Thus, this aspect of the present invention is directed to a programmed product, comprising signal-bearing media tangibly embodying a program of machine-readable

instructions executable by a digital data processor incorporating the CPU 1211 and hardware above, to perform the method of the invention.

This signal-bearing media may include, for example, a RAM contained within the CPU 1211, as represented by the fast-access storage for example. Alternatively, the instructions may be contained in another signal-bearing media, such as a magnetic data storage diskette 1300 (Figure 13), directly or indirectly accessible by the CPU 1211.

Whether contained in the diskette 1300, the computer/CPU 1211, or elsewhere, the instructions may be stored on a variety of machine-readable data storage media, such as DASD storage (e.g., a conventional "hard drive" or a RAID array), magnetic tape, electronic read-only memory (e.g., ROM, EPROM, or EEPROM), an optical storage device (e.g. CD-ROM, WORM, DVD, digital optical tape, etc.), paper "punch" cards, or other suitable signal-bearing media including transmission media such as digital and analog and communication links and wireless. In an illustrative embodiment of the invention, the machine-readable instructions may comprise software object code.

While the invention has been described in terms of a single preferred embodiment, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

Further, it is noted that, Applicants' intent is to encompass equivalents of all claim elements, even if amended later during prosecution.